PSZ 19:16 (Pind. 1/07)

UNIVERSITI TEKNOLOGI MALAYSIA

DECLARATION OF THESIS / UNDERGRADUATE PROJECT PAPER AND COPYRIGHT

Author's full name : **ILI SYAZANA BINTI IBRAHIM**

Date of birth : **10th JULY 1986**

Title : **USB DATA TRANSFER (HOST) FOR ALTERA DE2 BOARD**

Academic Session: **2008/2009**

I declare that this thesis is classified as :

☐	**CONFIDENTIAL**	(Contains confidential information under the Official Secret Act 1972)*
☐	**RESTRICTED**	(Contains restricted information as specified by the organisation where research was done)*
☑	**OPEN ACCESS**	I agree that my thesis to be published as online open access (full text)

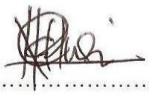
I acknowledged that Universiti Teknologi Malaysia reserves the right as follows :

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

	Certified by :
SIGNATURE	SIGNATURE OF SUPERVISOR
860710-87-5056	DR. NASIR SHAIKH HUSIN
(NEW IC NO. /~~PASSPORT NO.~~)	NAME OF SUPERVISOR
Date : 13th MAY 2009	Date :13th MAY 2009

NOTES : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organisation with period and reasons for confidentiality or restriction.

"I hereby declare that I have read this thesis and in my opinion this report has fulfills the scope and quality for the award of Degree of Engineering (Computer.)"



Signature : .....

Supervisor name : Dr. Nasir Shaikh Husin

Date : MAY 2009

USB DATA TRANSFER (HOST) FOR ALTERA DE2 BOARD

ILI SYAZANA BINTI IBRAHIM

A report in partial fulfilment of the
requirements for the award of the
degree of Bachelor of Engineering
(Computer)

Faculty of Electrical
Engineering
Universiti Teknologi Malaysia

May 2009

DECLARATION

I declare that this thesis entitled "USB Data Transfer (Host) for Altera DE2 Board" is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and not to concurrently submit in candidature of any other degree.

Signature : .....

Name : ILI SYAZANA BINTI IBRAHIM

Date : 14th MAY 2009

ACKNOWLEDGMENTS

Alhamdulillah I manage to complete my final year project within the time given. I am so blessed to Allah who gave me tremendous courage, strength and spirit to face all the obstacles all this while.

First and foremost, I would like to express my gratitude to my supervisor, Dr. Nasir Shaikh Husin for his advice and guidance given throughout the progress of this project.

I would like to thank valued phd student, Mr. Usman Ullah Sheikh for helping a lot in order to complete this project.

My appreciation goes to my family, who has always been there for me, to support and encourage me all these years. Thank you for the love and emotional support that they have given to me.

Finally, I would like to thank my fellow friends and anyone who is involved directly or indirectly with this project. My greatest thanks for all of you and may Allah bless you. Thank you.

Dedicated in thankful appreciation,
For support and care

To my beloved mother and father,
Sahariah and Ibrahim,

My understanding brothers and sisters,
Helmi, Liyana, Hazwan, Eleina and Amalina

My wonderful friends
My supportive lecturers

ABSTRACT

In this modern technology world, USB has become the most common serial bus interface between a device and a PC. GUI on the other hand is a way for human to interact with the computers that use windows, icons, and menus which can be manipulated by a mouse. This project is the combination of both USB and GUI to implement the USB applications using Altera DE2 board (NIOS II system). The GUI is a tool to connect the host computer and the NIOS II system using windows application. The objectives of this project are to enable the USB communication between the CPU and the NIOS II system and to design a GUI that can capture the data from the CPU and then transfer back the data to the CPU. The GUI created can sort the characters and can transfer bitmap image using USB as the medium to communicate between the host computer and the GUI. Based on the result obtained, it shows that the communication is successful as data can be captured and transferred.

ABSTRAK

Dalam dunia yang serba canggih ini, USB merupakan pengataramuka bas sesiri yang paling banyak digunapakai sebagai medium penghantaran data antara sesuatu alat dan CPU. GUI pula merupakan cara bagi manusia berkomunkasi dengan komputer menggunakan tetingkap, ikon, dan menu yang dimanipulasikan menggunakan tetikus. Projek ini adalah gabungan diantara kedua-dua GUI dan USB untuk membolehkan aplikasi USB menggunakan *ALTERA DE2 Development Board* (sistem NIOS II) dapat dijalankan. GUI digunakan sebagai medium alat untuk menghubungkan CPU dan sistem NIOS II menguunakan applikasi tetingkap. Obejktif projek ini adalah untuk membolehkan komunikasi diantara CPU dan sistem NIOS II dan untuk merekacipta GUI yang boleh menangkap data dari CPU dan kemudian menghantar kembali data tersebut ke CPU. GUI yang telah direkacipta boleh menyusun aksara secara sistematik berpandukan kepada kod ASCII dan juga boleh menghantar gambar bitmap dengan menjadikan USB sebagai medium untuk berkomunikasi. Berdasarkan keputusan yang diperoleh, komunikasi diantara CPU dan GUI adalah berjaya kerana data dapat ditangkap dan dihantar semula ke CPU.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	DECLARATION	ii
	ACKNOWLEDGEMENT	iii
	DEDICATION	iv
	ABSTRACT	v
	ABSTRAK	vi
	TABLE OF CONTENTS	vii
	LIST OF TABLES	ix
	LIST OF FIGURES	x
	LIST OF ABBREVIATIONS	xii
	LIST OF APPENDIX	xiii
1	**INTRODUCTION**	**1**
	1.1 Background	1
	1.1.1 Universal Serial Bus (USB)	1
	1.1.2 Graphical User Interface (GUI)	2
	1.2 Objectives	3
	1.3 Scope	4
	1.4 Outlined of thesis	4
2	**LITERATURE REVIEW**	**5**
	2.1 Introduction	5
	2.2 ALTERA DE2 Development Board	5
	2.3 USB Host and Device	8

	2.4	Coding for Serial Port Communication	11
3	**METHODOLOGY**		**13**
	3.1	Introduction	13
	3.2	Hardware	13
	3.3	Firmware	14
	3.4	GUI	15
		3.4.1 Microsoft Visual C++	16
		3.4.2 Microsoft Window Bitmap	17
		3.4.3 GUI for Sorting Characters	20
		3.4.3.1 Introduction	20
		3.4.3.2 The GUI Coding	22
		3.4.4 GUI for Sending Bitmap Image	23
		3.4.4.1 Introduction	23
		3.4.4.2 The GUI Coding	24
4	**RESULTS AND DISCUSSION**		**28**
	4.1	Sorting Characters	28
	4.2	Bitmap Transfer	31
5	**CONCLUSION AND RECOMENDATIONS**		**38**
	5.1	Conclusion	38
	5.2	Problems Encountered	38
	5.3	Recommendations	39
REFERENCES			**40**
APPENDIX A			**41**
APPENDIX B			**61**

LIST OF TABLES

TABLE NO.	TITLE	PAGE
3.1	Input Output (I/O) Addressing	14
3.2	Blocks of Data for BMP File	17

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
2.1	The diagram of the Altera DE2 Phillips Development Board	6
2.2	Schematic Diagram of Altera DE2 Board	6
2.3	Block Diagram of Altera DE2 Board	7
2.4	USB (ISP1362) Host and Device Schematic	9
2.5	Connection between the Three Register with ISP 1362 Chip	10
2.6	Snippet Code for Send Button (Serial Port)	11
2.7	Snippet Code for Receive Process OnRx (Part 1)	11
2.8	Snippet Code for Receive Process OnRx (Part 2)	12
3.1	Firmware Structure of the Device Controller of the ISP1362	15
3.2	Bitmap File Headers	18
3.3	Bitmap Info Headers	19
3.4	Workflow of the GUI for Sorting Characters	21
3.5	Snippet Code for Send Button	22
3.6	Snippet Code for Exit Button	23
3.7	Workflow for Sending Bitmap Image	24
3.8	Snippet Code for Load Button (Bitmap Image)	25
3.9	Snippet Code for Send Button (Bitmap Image)	26
3.10	Snippet Code for Exit Button(Bitmap Image)	27
4.1	Initial Condition of GUI for Sorting Characters	28
4.2	GUI After Entering the Characters	.29

4.3	GUI Contains the Sorted Characters	29
4.4	GUI with Numbers and Alphabets	30
4.5	GUI after Being Sorted	30
4.6	Initial Condition of GUI for Bitmap Image	31
4.7	GUI After Loading the Bitmap Image	32
4.8	GUI After the Image is Sent	32
4.9	25 Pixels Bitmap Image	33
4.10	64 Pixels Bitmap Image	34
4.11	144 Pixels Bitmap Image	34
4.12	14400 Pixels Bitmap Image	35
4.13	144 Pixels Bitmap Image (Part 2)	35
4.14	120x120 Pixels Bitmap Image(Part 2)	36
4.15	13x13 Pixels Bitmap Image	36
4.16	20x20 Pixels Bitmap Image	37

LIST OF ABBREVIATIONS

USB	-	Universal Serial Bus
GUI	-	Graphical User Interface
CPU	-	Central Processing Unit
DMA		Direct Memory Access
PIO		Programmed Input Output
HC		Host Controller
DC		Device Controller
DIB		Device Independence Bitmap
API		Application Programming Interface
RAM		Random Access Memory
FPGA		Field Programmable Gate Array

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Reference Coding for Sending Bitmap Image	41
B	Reference Coding for Sorting Characters	61

CHAPTER 1

INTRODUCTION

1.1 Background

1.1.1 Universal Serial Bus

USB is also known as Universal Serial Bus which is defined as a serial bus standard to interface devices to computer. It was designed to allow many peripherals to be connected using a single standardized interface socket and to improve the plug-and-play capabilities. This can be done by allowing devices to be connected and disconnected without rebooting the computer or turning off the device.

Universal Serial Bus (USB) is a set of connectivity specifications developed by Intel in collaboration with industry leaders. USB allows high-speed, easy connection of peripherals to a PC. When plugged in, everything configures automatically. USB is the most successful interconnect in the history of personal computing and has migrated into consumer electronics (CE) and mobile products.

USB offers several benefits such as low cost, expandability, auto-configuration, hot-plugging and outstanding performance. It also provides power to the bus, enabling many peripherals to operate without the added need for an AC power adapter. Since USB is a standard, third-party hardware is identical for all platforms; only differing in the software required for a specific OS.

The full speed of the USB nowadays is 2.0 M bit/s. USB can act as USB host interface or device interface depending on the application requirements. The most common use of the USB is as the host interface. The data can also be transferred using serial port communication, but this is slow compare to transferring using the USB.

1.1.2 Graphical User Interface(GUI)

GUI is the short form of graphical user interface. it is a type of user interface which allows people to interact with electronic devices such as computers. A GUI usually offers graphical icons, and visual indicators, as opposed to test-based interface, typed command label, or text navigation to fully respect the information and actions available to a user. These actions are usually performed through direct manipulation of the graphical elements

A GUI uses a combination of the technologies and devices to provide a platform the user can interact with, for the task of gathering and producing information. It is a way of communicating with a computer that uses visual feedback to the users as mush as possible.

Features of GUI includes the use of icons to represent commands and options, pull down menus that appears when called for and disappear when no longer needed, and the use of mouse to move pointer around the screen. By pointing to the appropriate icons or menu items and clicking a mouse button, various commands can be activated. It is possible to use the mouse in drawing programs.

Today's major operating systems provide a graphical user interface. Applications typically use the elements of the GUI that come with the operating system and add their own graphical user interface and ideas. With the increasing use of multimedia as part of the GUI, sound, voice, motion video and virtual reality interfaces seem likely to become part of the GUI for many applications. A system's graphical user interface along with its input devices is sometimes referred to as its "look-and-feel".

When creating an application, many object oriented tools exist that facilitate writing a graphical user interface. Each GUI element is defined as a class widget from which can create object instances for the application. The prepackaged methods that an object will use can be coded or modified to respond to user stimuli.

1.2 Objectives

The objective of this project is to enable the USB communication between CPU (the host computer) with the Altera DE2 Phillips Development Board. This communication can be done by designing a GUI (Graphical User Interface) to capture data from the CPU processes the data and later transfer back the data to the CPU using USB host.

1.3 Scope

There is only one scope for this project that has been outlined. The scope is to produce a working program that interfaces between the host computer and the hardware designed for it (half duplex communication) using USB.

1.4 Outline of Thesis.

This thesis basically contained four chapters. The first chapter is mostly about the background of the project. It discusses the objectives and the scope of the project. It gives a clear overview on what this project is about. In Chapter 2, it consists if theories and literature review for the project. It includes all the information about the hardware, the Altera DE2 Board, software and the GUI.

Chapter 3 will discuss on the methodology of the hardware and the software implementation of this project. All the results and discussions will be presented in Chapter 4. The last chapter which is Chapter 5 is about the conclusion and future recommendation to enhance this project.

CHAPTER 2

LITTERATURE REVIEW

2.1 Introduction

This chapter includes the study of the USB as a host interface and device interface, the overall view of the Altera DE2 Phillips Development Board and the GUI used.

2.2 Altera DE2 Development Board

This project uses the Altera DE2 Phillips Development Board as the medium to transfer and to process the data. The figure is shown in Figure 2.1. It depicts the layout of the board and indicates the location of the connectors and key components.

To get a better overview on the connection of the components in the DE2 Board, Figure 2.2 shows the schematic diagram of the board

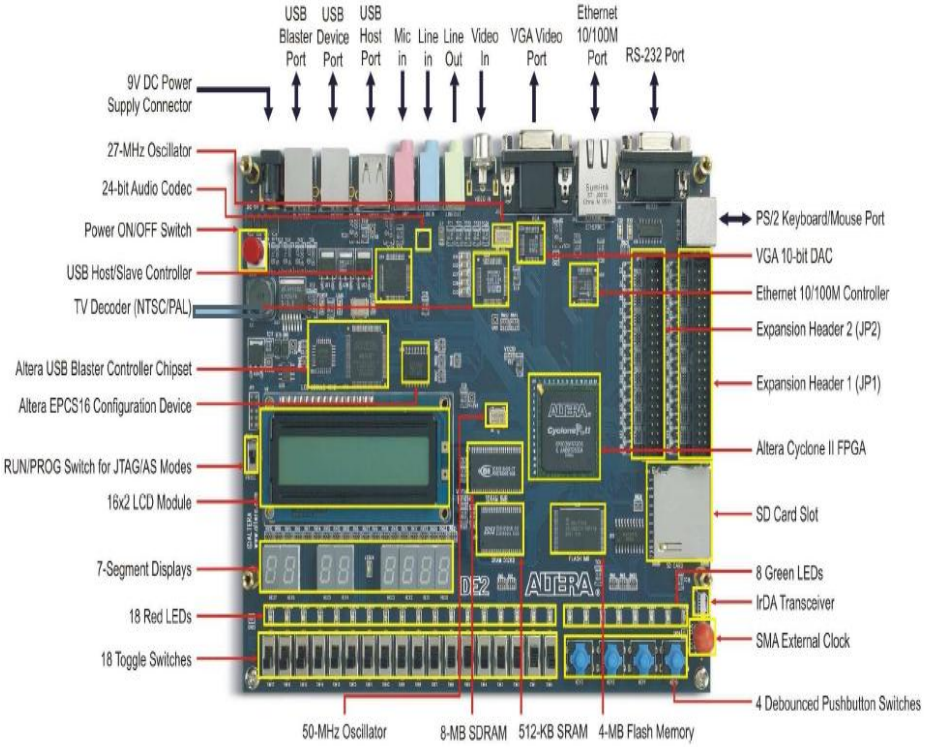


Figure 2.1: The diagram of the Altera DE2 Phillips Development Board

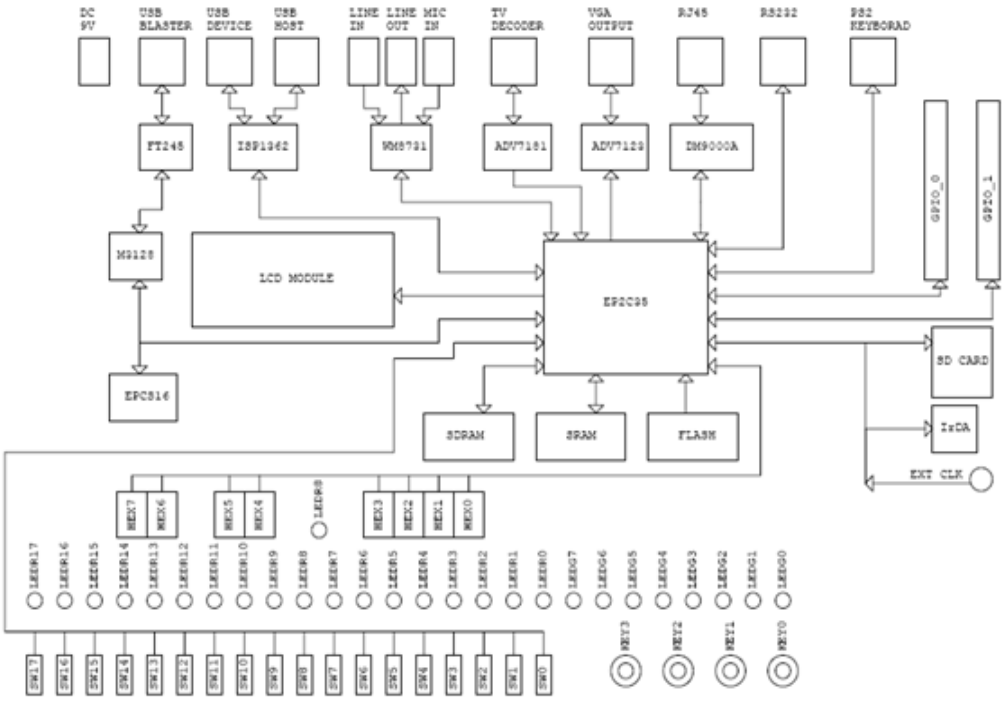


Figure 2.2 : Schematic Diagram of Altera DE2 Board

The maximum flexibility can be provided by connecting all of the connections on the board through the Cyclone II FPGA device. The FPGA can be configured to implement any design. Figure 2.3 shows the block diagram of the board.

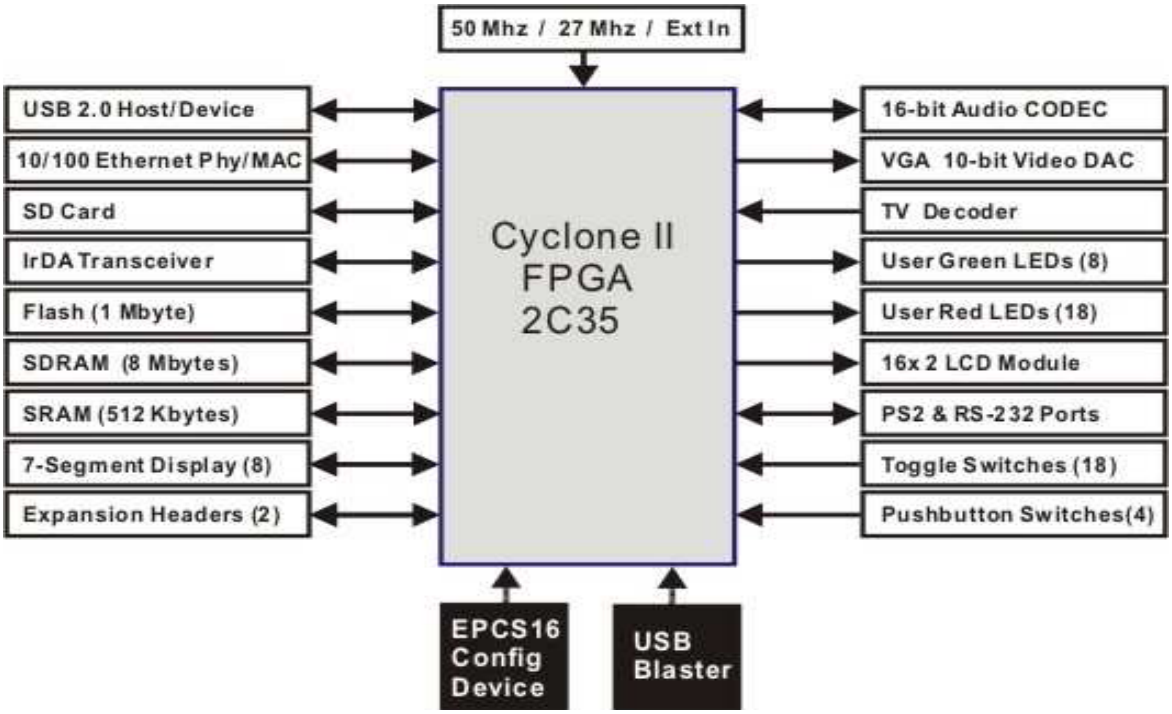


Figure 2.3: Block Diagram of Altera DE2 Board

For this project, the USB Host/Slave controller is used with the USB blaster to enable the transfer of data. This USB complies fully with the Universal Specification Rev 2.0. It can support data transfer at full-speed and also low-speed. The most importing thing is that it supports both USB host and USB device which in this project, USB device will be used..

Most USB applications and products operate as USB devices, rather than USB hosts. This project will operates base on USB device interfaces. The USB Host/Slave controller also provides a high-speed parallel interface to most available processors; supports NIOS II with a Terasic driver. Lastly, it can support

Programmed I/O (PIO) and Direct Memory Access (DMA) and which in this case, PIO will be used.

2.3 USB Host and Device

The DE2 board provides both USB host and device interfaces using the Philips ISP1362 single-chip USB controller. The host and device controllers are compliant with the Universal Serial Bus Specification Rev. 2.0, supporting data transfer at full-speed (12 Mbit/s) and low-speed (1.5 Mbit/s).The functionality of the ISP1362 can be accessed using a group of registers and two buffer memory areas. One buffer memory area is for the HC while the other is for the DC. Using PIO mode, registers can be accessed. However for buffer memory, it can be accessed using both PIO and direct memory access modes.

In general, there are 16 endpoints provided for the DC in the ISP 1362 for the USB device implementation. Each endpoint can be allocated RAM space in the on-chip ping pong buffer RAM. The endpoints are: endpoint 0 (control IN and OUT) and 14 configurable endpoints. This can be individually defined as interrupt, bulk or isochronous-IN or OUT. Each enabled endpoints has an associated buffer memory which can be accessed by either using the PIO or DMA mode.

Functions and registers of DC are accessed using the command, which consists of a command code. It is followed by optional data bytes (read or write action). The complete access consists of two phases.

The first is the command phase. When the address pin A0=HIGH, the DC interprets the data on the lower byte of the bus (bits D7 to D0) as a command code. Commands will be executed immediately if contains no data phase. The second part is optional which is data phase. When pin A0=LOW, the DC transfers the data on

the bus to or from a register or endpoint buffer memory. In case of multi-byte registers, the least significant byte or word are accessed first. Figure 2.4 shows the schematic diagram of the USB circuitry and the pin assigned.

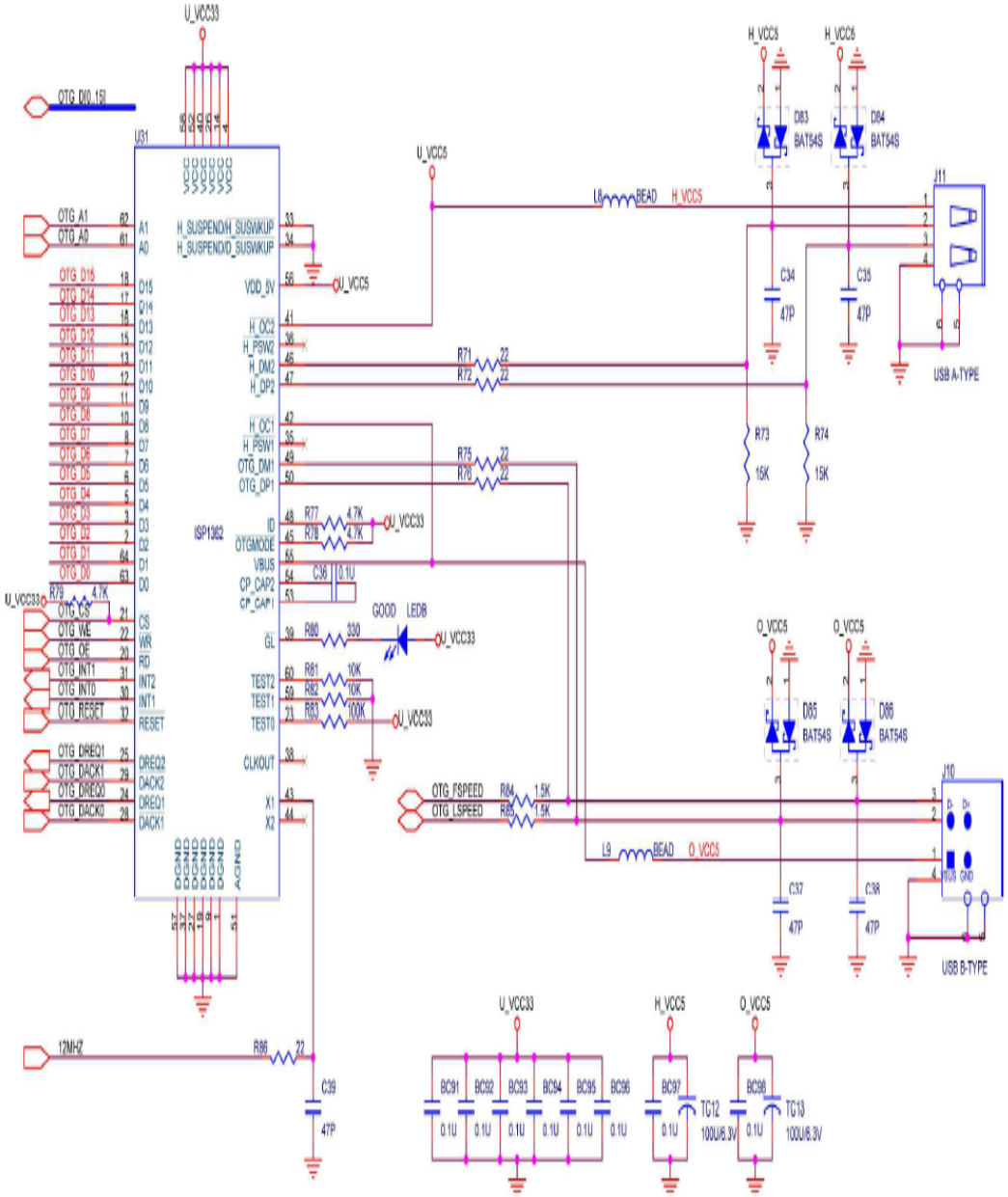


Figure 2.4: USB (ISP1362) Host and Device Schematic.

There are three groups of register in ISP1362 that control the function of the chip. There are OTG (On the Go) controller register, device controller register and

host controller register. This project uses the device controller register instead of the host controller register. The device controller is a cored based on Phillips device controller which is a full speed USB interface device. Figure 2.5 shows the three connections between the register and the ISP1362 chip. In the device mode, the ISP1362 acts as a USB device.

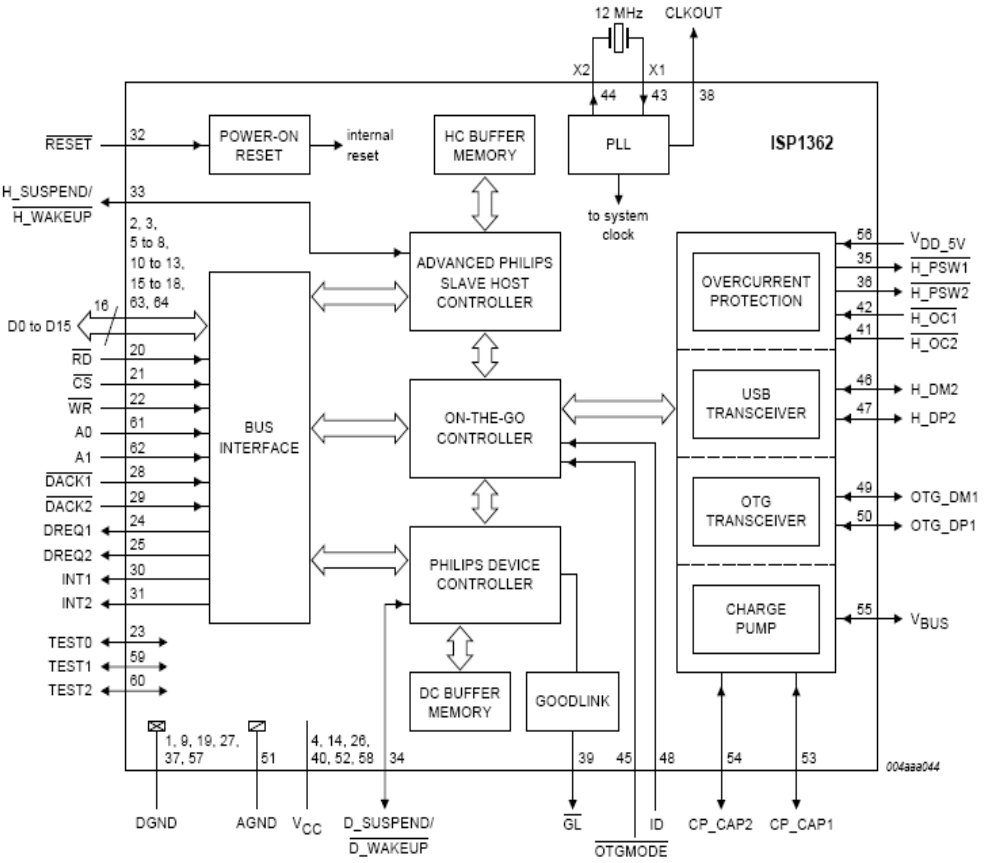


Figure2.5: Connection between the three register with ISP 1362 chip

2.4 Coding for Serial Port Communication

For serial port communication, there are some differences compared to the coding using the USB communication.

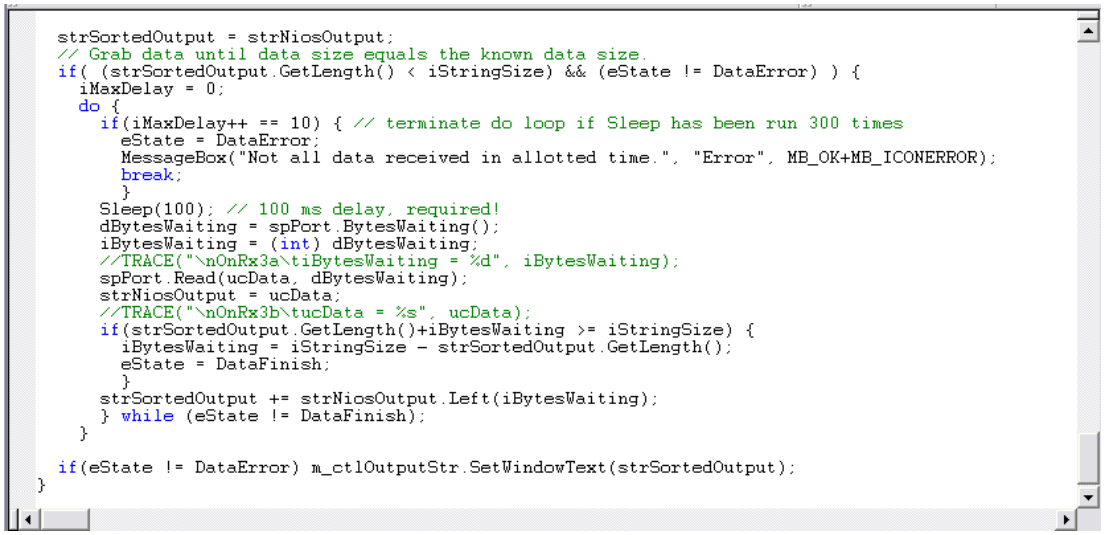
```
void CSerialNiosDlg::OnSend()
{
    // TODO: Add your control notification handler code here
  CString strNiosInput;

  /*void Open(int nPort, DWORD dwBaud=9600, Parity parity=NoParity, BYTE DataBits=8,
              StopBits stopbits=OneStopBit, FlowControl fc=NoFlowControl,
              BOOL bOverlapped=false);*/
  spPort.Open(1, 115200, CSerialPort::NoParity, 8, CSerialPort::OneStopBit,
              CSerialPort::NoFlowControl, false);

  m_ctlInputStr.GetWindowText(strNiosInput);
  iStringSize = strNiosInput.GetLength();

  //DWORD Write(const void *lpBuf, DWORD dwCount);
  spPort.Write(strNiosInput, (DWORD) iStringSize);
  spPort.TransmitChar('\r'); // send CR to comply with scanf protocol in Nios/Nios II
  OnRx();

}
```

Figure 2.6: Snippet Code for Send Button (Serial Port)

In Figure 2.6, it shows the snippet code for the Send button for serial port communication using the NIOS system processor. For the send button, need to first include the command spPort.Open. This command is used to open the serial communication port. To capture the characters from the edit box is basically the same as the communication using the USB. To send the data to the host computer, command spPort.Write is used. This functions exactly the same as the command WritePort1 for the USB. To read the data form the host computer, a function called OnRx () need to be called. The functions are shown in Figure 2.7 below.

```
void CSerialNiosDlg::OnRx()
{
  CString strNiosOutput = "";
  CString strSortedOutput = "";
  unsigned char ucData[400] = "";
  int iMaxDelay = 0;
  enum State {NoData, DataBegin, DataFinish, DataError};
  State eState = NoData;
  DWORD dBytesWaiting;
  int iBytesWaiting;

  Sleep(100);  // wait 100 ms
  //DWORD BytesWaiting(); returns # of bytes waiting to be read from serial port
  dBytesWaiting = spPort.BytesWaiting();
  iBytesWaiting = (int) dBytesWaiting;
  //TRACE("\nOnRx1\tiBytesWaiting = %d", iBytesWaiting);
  //DWORD Read(void *lpBuf, DWORD dwCount);
  spPort.Read(ucData, dBytesWaiting);
  strNiosOutput = ucData;
  //TRACE("\nOnRx1a\tstrNiosOutput = %s", strNiosOutput);
```

Figure 2.7: Snippet Code for Receive Process OnRx (Part 1)



```
  strSortedOutput = strNiosOutput;
  // Grab data until data size equals the known data size.
  if( (strSortedOutput.GetLength() < iStringSize) && (eState != DataError) ) {
    iMaxDelay = 0;
    do {
      if(iMaxDelay++ == 10) { // terminate do loop if Sleep has been run 300 times
        eState = DataError;
        MessageBox("Not all data received in allotted time.", "Error", MB_OK+MB_ICONERROR);
        break;
        }
      Sleep(100); // 100 ms delay, required!
      dBytesWaiting = spPort.BytesWaiting();
      iBytesWaiting = (int) dBytesWaiting;
      //TRACE("\nOnRx3a\tiBytesWaiting = %d", iBytesWaiting);
      spPort.Read(ucData, dBytesWaiting);
      strNiosOutput = ucData;
      //TRACE("\nOnRx3b\tucData = %s", ucData);
      if(strSortedOutput.GetLength()+iBytesWaiting >= iStringSize) {
        iBytesWaiting = iStringSize - strSortedOutput.GetLength();
        eState = DataFinish;
        }
      strSortedOutput += strNiosOutput.Left(iBytesWaiting);
      } while (eState != DataFinish);
    }

  if(eState != DataError) m_ctlOutputStr.SetWindowText(strSortedOutput);
}
```

Figure 2.8: Snippet Code for Receive Process OnRx (Part 2)

The whole coding that is shown in Figure 2.7 which is the Part 1 and Figure 2.8 which is the Part 2 can be simplify by just using the ReadPort2 command for the USB communication. For the serial communication, sleep (100) is required but for the USB communication, this can be neglected. The reason why USB is used instead of serial communication is because the rate of transfer for USB is much faster than the rate for serial communication.

CHAPTER 3

METHODOLOGY

3.1 Introduction

In this project, there are three parts of design. The first part is the hardware, second is the firmware and the last part is the GUI (Graphical User Interface). The hardware is already exists and will be implemented on the Altera DE2 Phillips development board to enable the data transfer. A software program (in this case is refer to the firmware) need to be executed on the NIOS II processor in order to initialize the Phillips ISP 1362 chip, after connecting the Altera DE2 Phillips development board to a USB port on the host computer.

3.2 Hardware

This project uses the PIO (Programmed Input Output) mode instead of DMA (Direct Memory Access). This memory access is similar to the register access, except the data phase is variable in length. There are two modes of addressing mode which are Indirect Addressing and Direct Addressing. Both modes are used in this project to access the host controller buffer.

The Table 3.1 shows the pin control for the bus interface. HC stands for host controller while DC stands for device controller. When the CS (chip select) is low, A1 is high and A0 is low, the ISP1362 chip will activate the device controller data port with 16 bits data bus width. While when the CS is lo1, A1 is high and A0 is high, the chip is the device controller command port.

Table 3.1: Input Output (I/O) Addressing

$\overline{CS}$	A1	A0	Access	Data bus width	Description
L	L	L	R/W	16 bits	HC data port
L	L	H	W	16 bits	HC command port
L	H	L	R/W	16 bits	DC data port
L	H	H	W	16 bits	DC command port

The codes for the hardware is written all in verilog codes and using the Quartus II version 8.0 software.

3.3 Firmware

The second part is called firmware where all the codes are written in C code. This is where the NIOS CPU capture and processes the data transferred. The firmware consists of C codes to initialize the ISP Phillips 1362 chip. The codes indicate that the chip is in device controller mode, which is a full speed USB interface device with up to 14 configurable endpoints. This device controller can be access by both DMA and PIO, and in this project, PIO is been used. This firmware is fully interrupt driven. Figure 3.1 illustrates the structure of the firmware of device controller ISP1362.

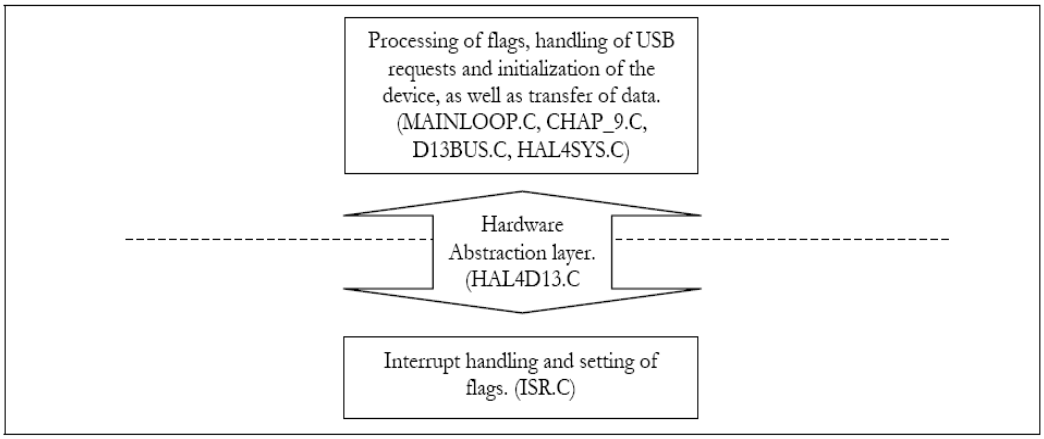

Processing of flags, handling of USB requests and initialization of the device, as well as transfer of data. (MAINLOOP.C, CHAP_9.C, D13BUS.C, HAL4SYS.C)
Hardware Abstraction layer. (HAL4D13.C
Interrupt handling and setting of flags. (ISR.C)

Figure 3.1: Firmware Structure of the Device Controller of the ISP1362

3.4 GUI

GUI (Graphical User Interface) is written fully in Visual C++ Version 6. This GUI is used to communicate with the DE2 Board. In order for the GUI to works, there are three steps that need to be taken. The first step is to copy the three files which are easyD12.dll, easyD12.h and easyD12.lib into the workspace folder. These folders are important as it contains the protocol for USB transfer using DE2 Board. Next, for the header file, must include '#include "EasyD12.h" ' before running the software. Last but not least, must include '#pragma comment (lib. "EasyD12.lib") '. This command is used to link the library EasyD12.lib to the GUI.

For this project, there are two GUIs that have been created to test the interfaces between the hardware and the DE2 board. The first one is GUI for sorting the integers and the second one is GUI for transferring bitmap image.

3.4.1 Microsoft Visual C++

Microsoft Visual C++ is known as MSVC, is an IDE (commercial integrated development environment). It is engineered for the C, C++, and C++/CLI programming languages by Microsoft. It has tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, and the DirectX API and the Microsoft.NET Framework.

Visual C++ 6.0 (commonly known as VC), was released in 1998 and it is popular until now and often used to maintain legacy projects. This project uses the version 6. In addition to the conventional graphical user interface applications, Visual C++ enables developers to build Web applications, smart client Window based applications and solutions for thin client and smart client mobile devices. C++ is the world's most popular system-level language, and Visual C++ gives developers a world class tool with which to build software.

There are a few reasons why it is chosen to develop the Graphical User Interface. The features such as the compile and build system feature, precompiled header files, "minimal rebuild" functionality and incremental link significantly shorten turn-around time to edit, compile and link the program, especially for large software projects. Because of the functionality that this software offers, Microsoft C++ is chosen to develop Graphic User Interface for this project.

Besides that, Visual C++ is the most powerful development tool available for Windows; excel at creating a wide array of applications including internet and web application. This includes the web server development. It also excels at creating server side components with ATL and creating rich user interfaces and robust applications with MFC. Lastly, it creates data centric applications with the Visual Database Tools and Universal Data Access.

The Microsoft Visual C++ is used to develop and load the image onto the NIOS II system for processing. A GUI is created to enable the image to be displayed after it is being transferred from the NIOS II system. The USB communication is implemented to send and receive the bitmap image in the Microsoft C++ environment. This communication allows the sending and receiving bitmaps pixel and data values between the Graphical User Interface and the NIOS II system.

3.4.2 Microsoft Windows Bitmap

Bitmap (BMP) is defined as a regular rectangular mesh of cells called pixels, which each pixel containing a colour value. Many graphical user interfaces use bitmaps in their built in graphics subsystem for example the Microsoft Windows. In un compressed BMP files, the pixels are characterized by only two parameters which are the number of pixels and the information content (colour depth) per pixel. The image pixels are stored with a colour depth of 1, 4, 8, 16, 24 or 32 bits per pixel. Images of 8 bits and fewer can be either gray scale or indexed colour. A typical BMP file usually contains the following blocks of data.

Table 3.2: Blocks of Data for BMP File

BMP File Header	Stores general information about the BMP file
Bitmap Information (DIB Header)	Stores detailed information about the bitmap images
Color Palette	Stores the definition of the colors being used for indexed color bitmap
Bitmap Data	Stores actual image, pixel by pixel.

The Bitmap file format changes as the Microsoft Windows changed. This changes causes there are about five or six versions of BMP files such as 1 bit (black

and white), 8 bit gray, 24 bit Red, Green and Blue, 8 bit indexed colour, 4 bit indexed colour, 32 bit Red, Green, Blue and Alpha.

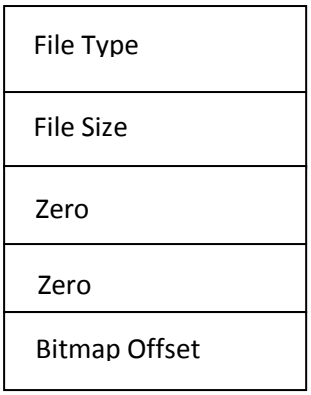


Figure 3.2.: Bitmap File Header

The bitmap files have a file header, a bitmap info header, colour table and image data. This is shown clearly in the Table 3.2 and Figure 3.2. For the file header, it contains 14 first bytes of the BMP file. The first two byte from the file header is the file type which is always equal to 4D42 in hexadecimal or 'BM'. For the size of the BMP file, it is located at the next four bytes. Next, the two bytes are reserved and always equal to zero. The last four bytes of the header give the offset to the image data. This values points to where in the file the image data begins. This value and the other four bytes values in the header, is an unsigned number (always positive).

Based of Figure 3.3, for the bitmap info header, it occupies the next 40 bytes. This bitmap info header begins with the size of the header and the value is always 40. Next in line are the width and height of the image data. This consists of the number of columns and rows. If the value of the height is a negative number, the image will be stored bottom-up which is the normal format for the BMP files. Next is the colour planes section. It refers to the number of colour planes used to store the bitmap. Usually the number of colour plane is 1. The information on bits per pixel shows the number of bits in a single pixel.

Header Size
Image Width
Image Height
Color Planes
Bits per pixel
Compression
Size of Bitmap
Horizontal Resolution
Vertical Resolution
Colors
Important Colors

Figure 3.3: Bitmap Info Header

The subsequent two fields deal with the image data compression. If the compression field is 0, it indicate no compression else if the compression field is equal to 1 it indicates for run length encoding compression. When the data is compressed, the size of bitmap fields gives the size of image data. When it is not compressed, it is equal to zero, and the software will calculate the size of the data.

The next two fields deal with the resolution of image data and the final two deal with the colours or gray shade in the image. The horizontal and vertical resolutions are expressed in pixel per meter. The colour fields help the software decipher the colour table discussed below. The colour fields states how many colours or gray shades are in the image. The important colour field states how many of the colours are important to the image.

The colour table comes after the header. A colour table is defined as a look up table that assign gray shade or colour to a number given in the image data. Colour table offers the opportunity to save space in image files and to match the image colour to a display or printing device. They do not play an important role in

the image processing routine for a gray scale images. There are four bytes in each colour table entry for the BMP colour table. The bytes are for the blue, green and red colour values. The fourth byte is padding and is always 0.

The last part of the BMP file is the image data. The data is stored row by row with padding on end of each row. The padding ensure the image rows are multiple by four. Multiple of four numbers, are just like the colour table, make it easier to read blocks and keep track of addresses.

In order to create new bitmap display for this project, the bitmap header and bitmap info header are cloned.

3.4.3 GUI for Sorting Characters

3.4.3.1 Introduction

The integers can be sorted by using the GUI designed. The GUI communicates with the DE2 board. The Figure 3.4 gives an idea about the route on how the GUI for sorting the integers works.

First, by clicking the Sort button in the window panel causes the host computer to send a particular USB packet to the DE2 board. The packet will be received by the NIOS II processor which will sort the strings sent. The sorted string can be seen in the console window of NIOS IDE 8.0 and the window panel (GUI).

When the exit button is click, the host computer sends a different USB packet to the board which causes the NIOS II processor to end the program and the window panel will be closed. The characters will be sorted based on the ASCII codes.

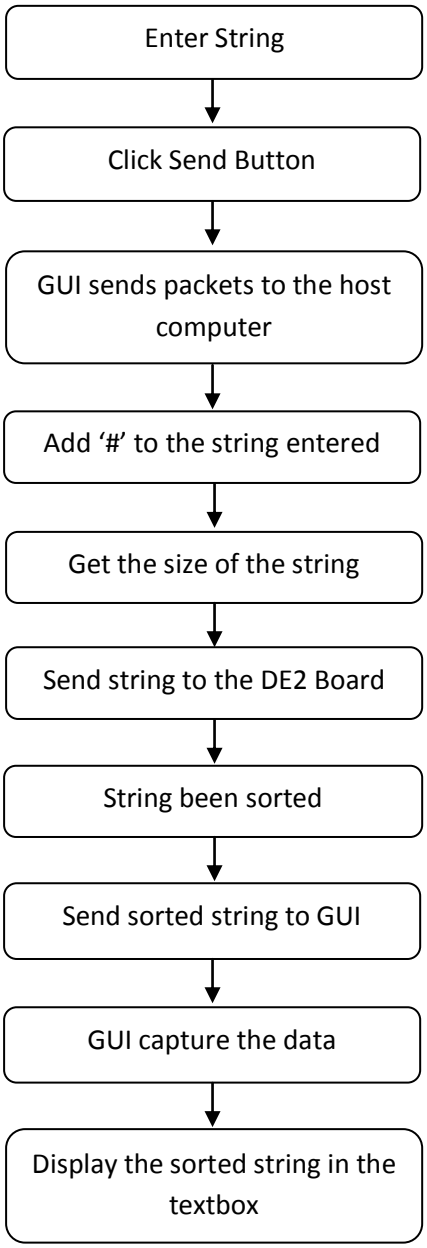

Enter String
Click Send Button
GUI sends packets to the host computer
Add '#' to the string entered
Get the size of the string
Send string to the DE2 Board
String been sorted
Send sorted string to GUI
GUI capture the data
Display the sorted string in the textbox

Figure 3.4: Workflow of the GUI for Sorting Characters

3.4.2.2 The GUI Coding

From the GUI, there are two buttons that need to be created which are the Send button and the Exit button. For the send button, the coding is shown below in the Figure 3.5.

```
void CSerialNiosDlg::OnSend()
{
    CString strNiosInput;
    m_ctlInputStr.GetWindowText(strNiosInput);
    strNiosInput += "#";

    iStringSize = strNiosInput.GetLength();

    BYTE *p = (BYTE*) strNiosInput.GetBuffer(0);
    WritePort1( (BYTE *) p, iStringSize);

    strNiosInput.ReleaseBuffer();

    char *pBuffer = new char [iStringSize + 1];

    ReadPort2( (BYTE *) pBuffer, iStringSize);

    pBuffer[iStringSize] = '\0';

    m_ctlOutputStr.SetWindowText(pBuffer);

    delete[] pBuffer;
```

Figure 3.5: Snippet Code for Send Button

Based on the coding, m_ctlInputStr.GetWindowText (strNiosInput) is basically a command to get and read the inputs that have been entered by the user. m_ctlInput is the name of the textbox that contains the input. strNiosInput is the name of the string. The command strNiosInput += "#" is to indicate that if a string is entered, character '#' will be added to that string.

GetBuffer(0) is the location of the string in the buffer. For example the strings 'try'. The alphabet't' is he location 0, alphabet 'r' is the location 1 and the alphabet 'y; is the location 3. The WritePort1 command is used to send the strings entered to the hardware to be sorted. The strNiosInput.ReleaseBuffer() is a

command to release the buffer. This is necessary to release the buffer to empty up the space in the memory.

For textbox, string size must be terminated with a space. That is why command [iStringSize + 1] is needed. To write back the sorted string on the edit box, command ReadPort2 is used.

```
void CSerialNiosDlg::OnExit()
{
    // TODO: Add your control notification handler code here
  spPort.Close();
  OnOK();
}
```

Figure 3.6: Snippet Code for Exit Button

For the Exit button, the coding is shown in Figure 3.6. Basically the command for exit is simple. The commands shown are to close the port for communication and exit the window screen.

3.4.4 GUI for Sending Bitmap Image

3.4.4.1 Introduction

The bitmap images can be sent to the DE2 board, and been transfer back to the GUI. The Load button will enable the users to browse and choose any bitmap

image to be loaded in the GUI. Then, the send button will send packets of the image to the DE2 board and the host computer will send the packets back to the GUI. The workflow is shown in Figure 3.7.

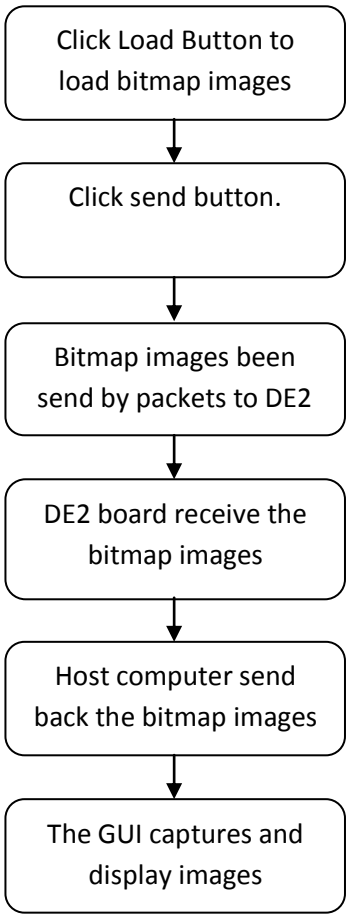

Click Load Button to load bitmap images
Click send button.
Bitmap images been send by packets to DE2
DE2 board receive the bitmap images
Host computer send back the bitmap images
The GUI captures and display images

Figure 3.7: Workflow for Sending Bitmap Image

3.4.4.2 The GUI Coding

There are three buttons that need to be created for this GUI design. The first button is Load BMP, then Send and the last button is Exit. For the Load BMP button, the coding is given in Figure 3.8.

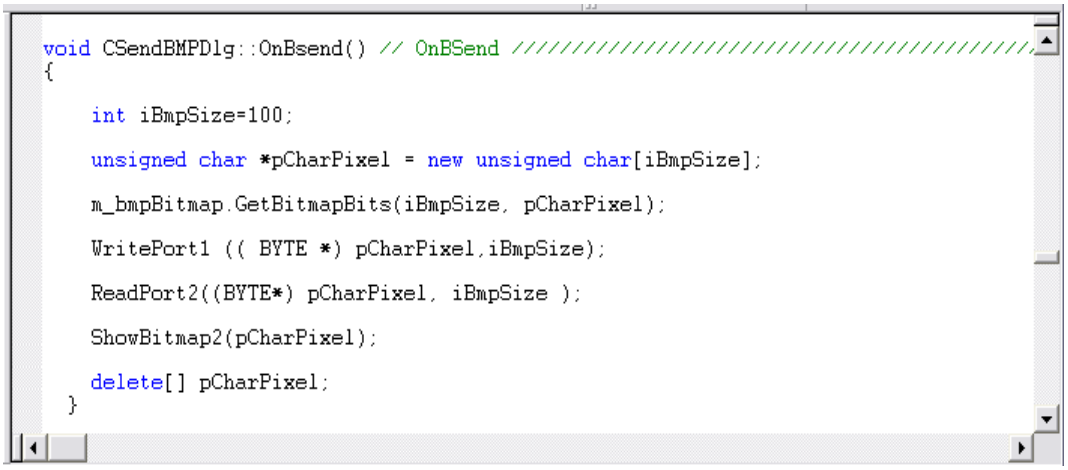
```
void CSendBMPDlg::OnBload()
{
    // TODO: Add your control notification handler code here
  // Build a filter for use in the File Open dialog
  static char BASED_CODE szFilter[] = "Bitmap Files (*.bmp)|*.bmp||";
  // Create the File Open dialog
  CFileDialog m_ldFile(TRUE, ".bmp", m_sBitmap, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
                       szFilter);

  // Show the File Open dialog and capture the result
  if (m_ldFile.DoModal() == IDOK) {
    // Get the filename selected
    m_sBitmap = m_ldFile.GetPathName();
    // Load the selected bitmap file
    HBITMAP hBitmap = (HBITMAP) ::LoadImage(AfxGetInstanceHandle(), m_sBitmap, IMAGE_BITMAP,
                       0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);
    // Repeat for clone image. We do this just to capture the BMFIH & BMFH header data!
    HBITMAP hBitmap2 = (HBITMAP) ::LoadImage(AfxGetInstanceHandle(), m_sBitmap, IMAGE_BITMAP,
                       0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);

    // Do we have a valid handle for the loaded image?
    if (hBitmap) {
      // Delete the current bitmap
      if ( m_bmpBitmap.DeleteObject() )
        // If there was a bitmap, detach it
        m_bmpBitmap.Detach();
      // Attach the currently loaded bitmap to the bitmap object
      m_bmpBitmap.Attach(hBitmap);
      // Create clone bitmap object, so that we can capture the BMFIH & BMFH header data!
      if ( m_bmpBitmap2.DeleteObject() ) m_bmpBitmap2.Detach();
      m_bmpBitmap2.Attach(hBitmap2);

      iFlagDataReceived = 0; // tell OnPaint not to redraw 2nd frame
      Invalidate();
      }
    }
}
```

Figure 3.8: Snippet Code for Load Button (Bitmap Image)

For this coding, the CFileDialog m_ldFile(TRUE, ".bmp", m_sBitmap, OFN_HIDEREADONLY |OFN_OVERWRITEPROMPT, szFilter) command is to create the file open dialog box. TRUE is used to construct a File Open dialog box. If FALSE, it is used to construct File Save As dialog box. ".bmp" is the default file extension. Since in this project, bitmap format is used, thus the default extension is ".bmp". This is automatically appended to the file name.

M_sBitmap is the initial filename that appear in the filename edit box. OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT is called flags. The combination of this two flags allows the dialog box to be customized. szFilter is a series of strings pair that specify filter that can be applied to the file.

Figure 3.9 illustrates the snippet code for send button for the bitmap image.



```
void CSendBMPDlg::OnBsend() // OnBSend ////////////////////////////////////////
{
    int iBmpSize=100;
    unsigned char *pCharPixel = new unsigned char[iBmpSize];
    m_bmpBitmap.GetBitmapBits(iBmpSize, pCharPixel);
    WritePort1 (( BYTE *) pCharPixel,iBmpSize);
    ReadPort2((BYTE*) pCharPixel, iBmpSize );
    ShowBitmap2(pCharPixel);
    delete[] pCharPixel;
  }
```

Figure 3.9: Snippet Code for Send Button (Bitmap Image)

The size of the bitmap image has been hardcoded which is 100 pixels. Since the data that need to be transferred is in byte, there is no conversion needed unlike the string for the sorted data. The pCharPixel is the bitmap image and it is in byte. m_bmpBitmap.GetBitmapBits(iBmpSize, pCharPixel) is used to get the bitmap's pixel value (captured data that has been loaded in the edit box).

To send the data to the hardware, the command used is similar to the sorted string which is WritePort1. For the command to send back the data to the GUI, Readport2 is used. ShowBitmap2(pCharPixel) is the command used to display the bitmap image in the edit box. This command is important as without this command, the images cannot be displayed..

For the Exit button, it is shown in Figure 3.10. Basically the command for exit is simple. The commands shown are to close the port for communication and exit the window screen.

```
void CSerialNiosDlg::OnExit()
{
    // TODO: Add your control notification handler code here
  spPort.Close();
  OnOK();
}
```

Figure 3.10: Snippet Code for Exit Button(Bitmap Image)

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Sorting Characters

Figure 4.1: Initial Condition of GUI for Sorting Characters

In the Figure 4.1 above, is the GUI for sorting the characters. This GUI consists of two buttons which are Send button and Exit button. This is the initial condition of the GUI before the user enters any characters.

Figure 4.2: GUI After Entering the Characters

This Figure 4.2 shows that the users enter characters “Selamat Datang ke UTM!!”. Then the Send button is clicked.

Figure 4.3: GUI Contains the Sorted Characters

The Figure 4.3 shows the result after the Send button is clicked. In the Sorted String edit box, it can be shown clearly that the string is already sorted. In hexadecimal, ‘a’ is represented as integer 61. The strings are sorted based on the ASCII code. Space is represented as integer 20 which is the smallest number of all the strings enter. That is the reason why it is placed in front of other strings. Character “!” is 21 in ASCII code and it is the second smallest number. It is placed after space. And other strings are sorted based on this method.

Figure 4.4: GUI with Numbers and Alphabets

If for example, integers is entered as well as shown in Figure 4.4, the outcome is the same as example before. The strings will be sorted based on ASCII code, by putting the smallest number in front first.

Figure 4.5: GUI after Being Sorted

This Figure 4.5 illustrates the result of sorting integers and alphabets. The characters entered are testing123. Number 1 in ASCII code is 48. And it is placed in front as it is the smallest number compared to all other strings. Alphabet‘t’ is the largest ASCII number, so it is placed at the back of the sorted string.

4.2 Sending Bitmap Image

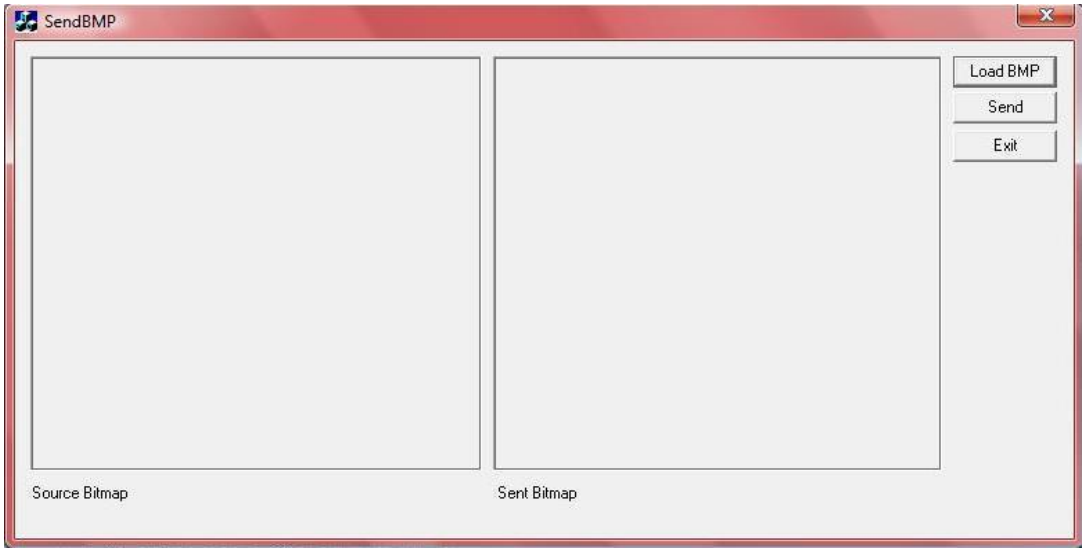


Figure 4.6: Initial Condition of GUI for Bitmap Image

In the Figure 4.6 demonstrate the GUI for sending bitmap image. The GUI consists of three buttons which are the Load BMP button, Send button and an exit button. This is the initial condition of the GUI before loading any bitmap image.

For this GUI, there is a load button and both of the spaces are edit box. The load button is used to load only bitmap files from any folder. User can browse through the folders is the computer before loading it to the GUI.

Figure 4.7 shows a bitmap image already been uploaded in the edit box named Source Bitmap. The image's size is 10x10 pixels.

Figure 4.7: GUI After Loading the Bitmap Image

Figure 4.8: GUI After the Image is Sent

After the Send button is clicked, the bitmap image will be transferred to the DE2 Board, and the will be transferred back as shown in the Figure 4.8. Since there is no modification is made, the image remain the same

Since the bitmap image is hardcoded as 100 pixels, only image with size 100 pixels or smaller then 100 pixels can be sent through the GUI. The example is shown in Figure 4.9. The size of the bitmap image is 5x5 pixels which equals to 25 pixels. For Figure 4.10, the size is 8x8 pixels which is equal to 64 pixels.

Based on Figure 4.11, some of the sent pixels seemed to be corrupted. The reason for this situation to occur is the value of the bitmap size is larger than 100. This image has 12 pixels height and 12 pixels width which the size of the image is 144. This value already exceeds the value that has been fixed.

The second example for this situation to occur can be depicted in Figure 4.12. For this figure, most of the sent images seemed to be corrupted. The actual size of this bitmap image is 120x120 which is equal to 14400. This value is beyond the limit that had already been hardcoded earlier. As the result, only 100 pixels can be sent while the others are corrupted.

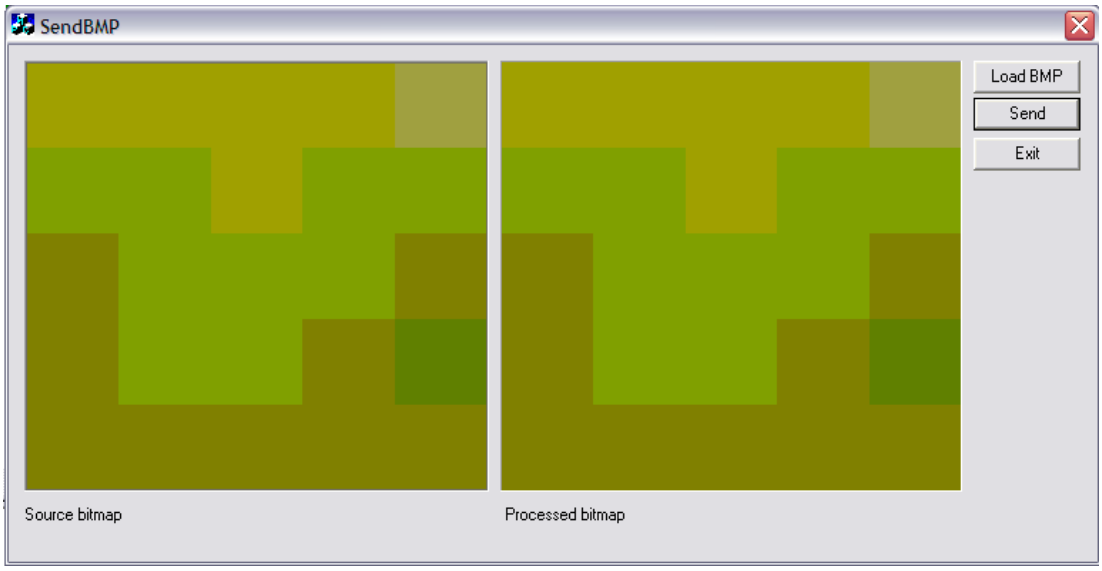


Figure 4.9:25 Pixels Bitmap Image

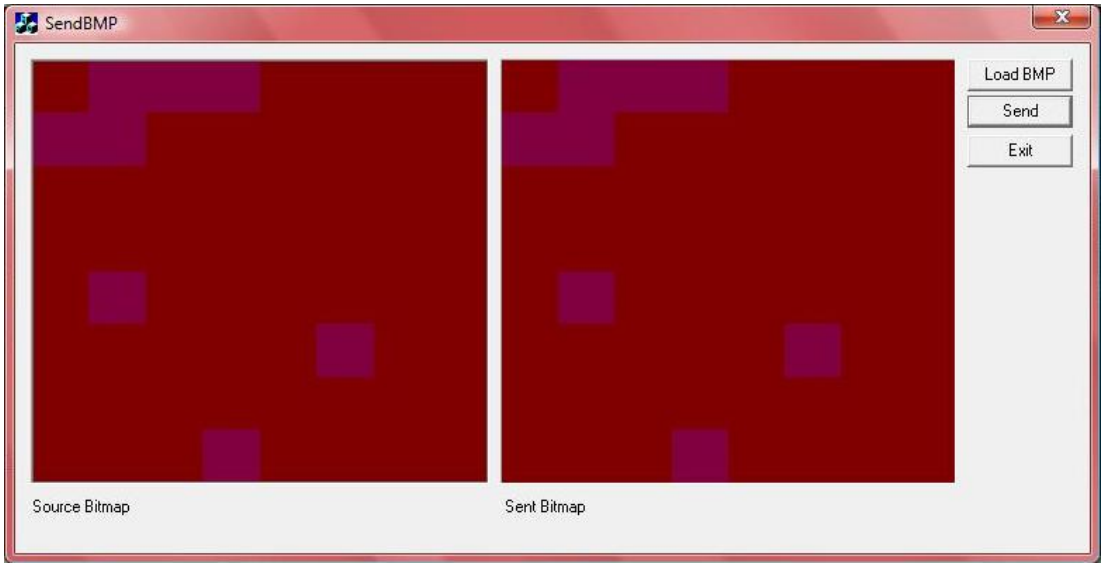


Figure 4.10:64 Pixels Bitmap Image

Figure 4.11:144 Pixels Bitmap Image

For the second example; the value of the bitmap size is changed to 144 pixels. Form Figure 4.13, the size of the bitmap image is 12x12 equals to exactly 144. When the image is sent to the NIOS II system, the sent bitmap is the same as the source bitmap indicating that there is no data corrupted.

For Figure 4.14, the images are not same as corruption of data occurred. This is because the value of 144 is smaller than 14400. Not all data can be transferred and the data maybe lost or corrupted.

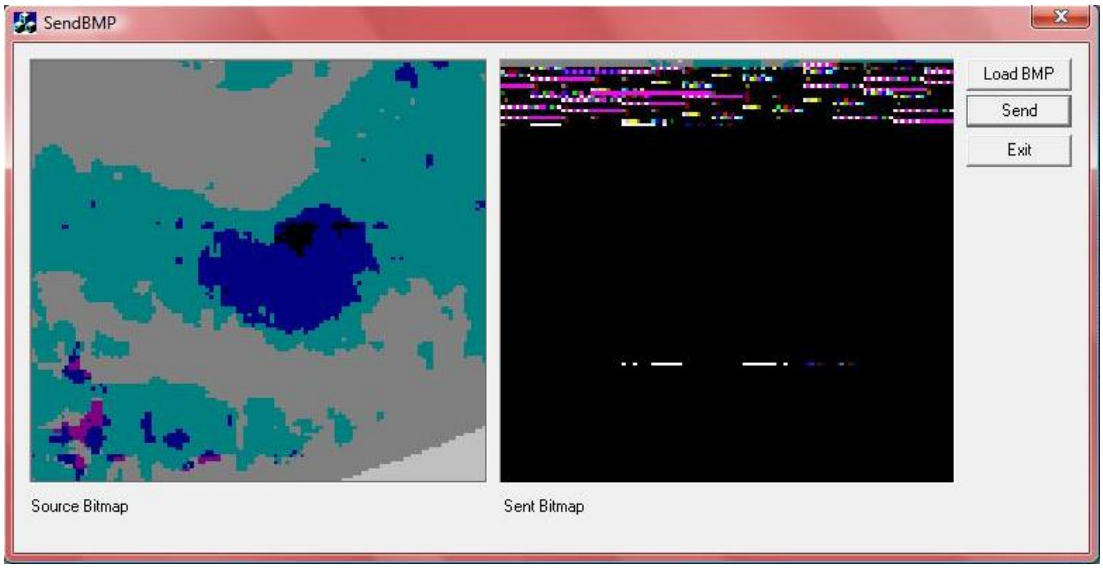


Figure 4.12:14400 Pixels Bitmap Image

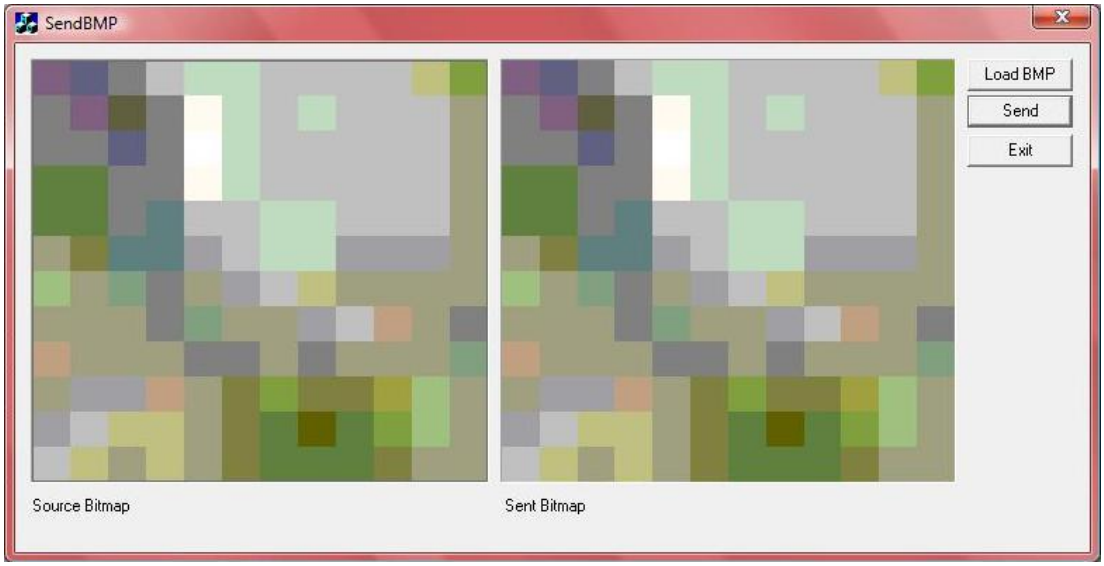


Figure 4.13: 144 Pixels Bitmap Image (Part 2)

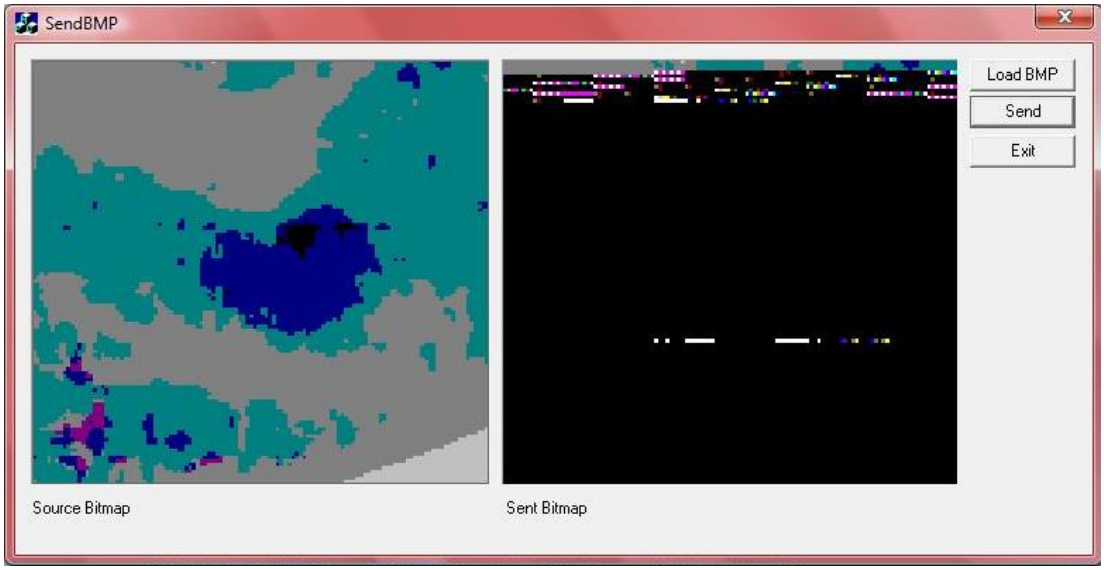


Figure 4.14: 120x120 Pixels Bitmap Image (Part 2)

For the next example, the data is hardcoded as 169 pixels. In Figure 4.15, the bitmap image's size is 169 pixels. This shows that all the data has been transferred as both of the images are the same.

Figure 4.15: 13x13 Pixels Bitmap Image

The last example, the size is hardcoded as 4000. This indicates that the height and width each is a value of 20 pixels. In Figure 4.16, the bitmap image is

equal to the hardcoded value. Thus, there will be no data lost during the processing stage.

Figure 4.16: 20x20 Pixels Bitmap Image

.

CHAPTER 5

CONCLUSION AND FUTURE RECOMENDATIONS

5.1 Conclusion

From the results that have been obtained, the communication between the DE2 board and the hardware is successful as both GUI are able to capture data and send back to the host computer. The objectives of this project are successfully fulfilled.

5.3 Problems Encountered

There are two major problems that have been encountered during the progress of this project. The first problem is regarding the firmware. The data transfer can only be done for eight or less characters only. If the number of character exceeds eight byte, the output will turned out to be wrong. The solution for this problem is doing some modification which is looping to enable more than eight characters to be transferred.

For the GUI part, there is a problem faced regarding the height and width of the bitmap image. The GUI cannot identify the height and width of images to solve this problem, the height and width of the bitmap image is hardcoded. This way the user cannot specify the actual height and width. Though the GUI woks, but it is not efficient. If the size of bitmap is hardcoded for example 100 pixels, and the user wants to load a bitmap image and the size is 120, the user need to change first the size in the coding. If the user refuses to do so, there will be an error as only 100 pixels are transferred while the other 20 pixels is considered corrupted.

5.2 Future Recommendations

To make this project works efficiently, the size of the bitmap image need not to be hardcoded every time users want to load bitmap image. The height and width can be specified and the user can load image without having to think about the maximum size that has been hardcoded.

Besides that, to enhance this project, image processing can be done and then been transferred back to the GUI. Image processing techniques such as median filtering, image wrapping, morphing and digital half toning that can be used to make this project more challenging.

REFERENCES

1. ALTERA Corporation (2006).*Embedded Programming Guide*
2. ALTERA Corporation (2006).*ISP 1362 Datasheet*
3. ALTERA Corporation (2006).*DE2 User Manual*
4. ALTERA Corporation (2006).*Embedded Programming Guide*
5. Chapman D. (1998). *Sams Teach Yourself Visual C++ 6 in 21 Days*. Indianapolis,Indiana, USA: SAMS
6. Don Anderson (1997), *USB System Architecture*", Addison-Wesley.
7. Ivor Horton. *Beginning Visual C++ Version 6 (paperback)*,Wrox

APPENDIX A

Reference Coding for Sending Bitmap Image

```
// SendBMPDlg.cpp : implementation file

#include "stdafx.h"

#include "SendBMP.h"

#include "SendBMPDlg.h"

#ifdef _DEBUG

#define new DEBUG_NEW

#undef THIS_FILE

static char THIS_FILE[] = __FILE__;

#endif


// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog

{

public:

        CAboutDlg();


// Dialog Data

        //{{AFX_DATA(CAboutDlg)

        enum { IDD = IDD_ABOUTBOX };

        //}}AFX_DATA

        // ClassWizard generated virtual function overrides

        //{{AFX_VIRTUAL(CAboutDlg)

        protected:

        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        //}}AFX_VIRTUAL
```

```
// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP(){
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
```

```
// CSendBMPDlg message handlers

BOOL CSendBMPDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	// TODO: Add extra initialization here
  return TRUE;  // return TRUE  unless you set the focus to a control
}
void CSendBMPDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
```

```
// CSendBMPDlg dialog


CSendBMPDlg::CSendBMPDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSendBMPDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSendBMPDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CSendBMPDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSendBMPDlg)
	DDX_Control(pDX, IDC_FRAME2, m_ctlFrame2);
	DDX_Control(pDX, IDC_FRAME, m_ctlFrame);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CSendBMPDlg, CDialog)
	//{{AFX_MSG_MAP(CSendBMPDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BEXIT, OnBexit)
	ON_BN_CLICKED(IDC_BLOAD, OnBload)
```

```
ON_BN_CLICKED(IDC_BSEND, OnBsend)
        ON_BN_CLICKED(IDC_BUTTON1, OnClear)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
BOOL CSendBMPDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.
        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                        strAboutMenu);
                }
        }
```

```
	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog

	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

  return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSendBMPDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
```

```
	void CSendBMPDlg::OnPaint()
{
	CDC *pdc = m_ctlFrame.GetDC(); // device context for painting

	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
```

```
            else
            {
                        CDialog::OnPaint();
    if (m_sBitmap != "") ShowBitmap(pdc);
    if (iFlagDataReceived) ShowBitmap2(NULL); // flag is stipulated in OnBload & OnRx
            }
  m_ctlFrame.ReleaseDC(pdc);
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSendBMPDlg::OnQueryDragIcon()
{
            return (HCURSOR) m_hIcon;
}

void CSendBMPDlg::OnBexit()
{
            // TODO: Add your control notification handler code here
  spPort.Close();
  OnOK();

}
```

```
        void CSendBMPDlg::OnBload()
{
        // TODO: Add your control notification handler code here
 // Build a filter for use in the File Open dialog
 static char BASED_CODE szFilter[] = "Bitmap Files (*.bmp)|*.bmp||";
 // Create the File Open dialog
 CFileDialog m_ldFile(TRUE, ".bmp", m_sBitmap, OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT,
                szFilter);

 // Show the File Open dialog and capture the result
 if (m_ldFile.DoModal() == IDOK) {
  // Get the filename selected
  m_sBitmap = m_ldFile.GetPathName();
  // Load the selected bitmap file
  HBITMAP hBitmap = (HBITMAP) ::LoadImage(AfxGetInstanceHandle(), m_sBitmap,
IMAGE_BITMAP,
                0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);
  // Repeat for clone image. We do this just to capture the BMFIH & BMFH header
data!
  HBITMAP hBitmap2 = (HBITMAP) ::LoadImage(AfxGetInstanceHandle(), m_sBitmap,
IMAGE_BITMAP,
                0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTION);
```

```
   // Do we have a valid handle for the loaded image?
   if (hBitmap) {
     // Delete the current bitmap
     if ( m_bmpBitmap.DeleteObject() )
       // If there was a bitmap, detach it
       m_bmpBitmap.Detach();
     // Attach the currently loaded bitmap to the bitmap object
     m_bmpBitmap.Attach(hBitmap);
     // Create clone bitmap object, so that we can capture the BMFIH & BMFH header
data!
     if ( m_bmpBitmap2.DeleteObject() ) m_bmpBitmap2.Detach();
     m_bmpBitmap2.Attach(hBitmap2);

     iFlagDataReceived = 0; // tell OnPaint not to redraw 2nd frame
     Invalidate();
     }
   }
}
// Do we have a valid handle for the loaded image?
   if (hBitmap) {
     // Delete the current bitmap
     if ( m_bmpBitmap.DeleteObject() )
       // If there was a bitmap, detach it
```

```
      m_bmpBitmap.Detach();
    // Attach the currently loaded bitmap to the bitmap object
    m_bmpBitmap.Attach(hBitmap);
    // Create clone bitmap object, so that we can capture the BMFIH & BMFH header
data!
    if ( m_bmpBitmap2.DeleteObject() ) m_bmpBitmap2.Detach();
    m_bmpBitmap2.Attach(hBitmap2);

    iFlagDataReceived = 0; // tell OnPaint not to redraw 2nd frame
    Invalidate();
    }
   }
}
void CSendBMPDlg::ShowBitmap(CDC *pdc)
{
  CRect lRect;
  // Get the loaded bitmap
  m_bmpBitmap.GetBitmap(&m_bmpBM);
  CDC dcMem;

// Create a device context to load the bitmap into
  dcMem.CreateCompatibleDC(pdc);
```

```
 // Select the bitmap into the compatible device context
 dcMem.SelectObject(m_bmpBitmap);
 // Get the display area available
 m_ctlFrame.GetClientRect(lRect);
 lRect.NormalizeRect();
 // Copy and resize the bitmap to the dialog window
 pdc->StretchBlt(0, 0, lRect.Width(), lRect.Height(), &dcMem, 0, 0,
m_bmpBM.bmWidth,
            m_bmpBM.bmHeight, SRCCOPY);
 //pdc->BitBlt(0, 0, lRect.Width(), lRect.Height(), &dcMem, 0, 0, SRCCOPY);
}

#include "EasyD12.H"
#pragma comment (lib, "EasyD12.lib")

void CSendBMPDlg::OnBsend() // OnBSend
{
        // TODO: Add your control notification handler code here
        int iBmpSize=100;
        unsigned char *pCharPixel = new unsigned char[iBmpSize];
```

```
WritePort1 (( BYTE *) pCharPixel,iBmpSize);



        /*void Open(int nPort, DWORD dwBaud=9600, Parity parity=NoParity, BYTE
DataBits=8,
        StopBits stopbits=OneStopBit, FlowControl fc=NoFlowControl,
        BOOL bOverlapped=false);*/
 // spPort.Open(1, 115200, CSerialPort::NoParity, 8, CSerialPort::OneStopBit,
      //    CSerialPort::NoFlowControl, false);


 /*TRACE("m_bmpBM.bmBits = %d\n", m_bmpBM.bmBits);
 TRACE("m_bmpBM.bmBitsPixel = %d\n", m_bmpBM.bmBitsPixel);
 TRACE("m_bmpBM.bmHeight = %d\n", m_bmpBM.bmHeight);
 TRACE("m_bmpBM.bmPlanes = %d\n", m_bmpBM.bmPlanes);
 TRACE("m_bmpBM.bmType = %d\n", m_bmpBM.bmType);
 TRACE("m_bmpBM.bmWidth = %d\n", m_bmpBM.bmWidth);
 TRACE("m_bmpBM.bmWidthBytes = %d\n", m_bmpBM.bmWidthBytes);*/


 m_bmpBitmap.GetBitmapBits(iBmpSize, pCharPixel);
        ReadPort2((BYTE*) pCharPixel, iBmpSize );
        //pCharPixel[strNiosInput.GetLength()] = '\0';
//      CString str(pCharPixel);
```

```
//spPort.Write( strNiosInput, (DWORD) strNiosInput.GetLength() );
   //spPort.TransmitChar('\r'); // send CR to comply with scanf protocol in Nios/Nios II
   // send pixel height
  //  strNiosInput.Format("%d", m_bmpBM.bmHeight);
   // spPort.Write( strNiosInput, strNiosInput.GetLength() );
    //spPort.TransmitChar('\r');
  //  int iSendSize = 32; // anything larger won't work!!!
  //  int iSendIteration = iBmpSize/iSendSize;
  //  int iSendRemainder = iBmpSize%iSendSize;
  //  int i;
  //  for(i=0; i<iSendIteration; i++) {
   // spPort.Write(&pCharPixel[iSendSize*i], (DWORD) iSendSize);
   //   Sleep(10); // required!!!

   // if(iSendRemainder) spPort.Write(&pCharPixel[iSendSize*i], (DWORD)
iSendRemainder);
   // MessageBox("Data sent.", "Note", MB_OK);
   //OnRx();
   // }
            ShowBitmap2(pCharPixel);
  delete[] pCharPixel;
  }
```

```
void CSendBMPDlg::OnRx()// OnRx
//////////////////////////////////////////////////////

{

 const int iBufSize = 400;

 unsigned char ucData[iBufSize] = ""; // size must be consistent!

 unsigned char ucDataSorted[76800] = ""; // 320x240 bitmap size for 8-bit images

 int iDataSize;

 int iMaxDelay = 0;

 int iDataError = 0;

 int iDataFinish = 0;

 DWORD dBytesWaiting;

 int iBytesWaiting;


 Sleep(100);  // wait 100 ms

 //DWORD BytesWaiting(); returns # of bytes waiting to be read from serial port

 dBytesWaiting = spPort.BytesWaiting();

 if( (int) dBytesWaiting > iBufSize ) dBytesWaiting = (DWORD) iBufSize;

 //DWORD Read(void *lpBuf, DWORD dwCount);

 spPort.Read(ucDataSorted, dBytesWaiting);

 //TRACE("\nOnRx1a\tucDataSorted = %s", ucDataSorted);
```

```
iDataSize = (int) dBytesWaiting;
 //TRACE("\nOnRx2\tiDataSize = %d", iDataSize);

 // Grab data until data size equals the known data size.
 int iBmpSize = m_bmpBM.bmWidthBytes*m_bmpBM.bmHeight;
 if(iDataSize < iBmpSize) {
  iMaxDelay = 0;
  do {
   if(iMaxDelay++ == 300) {
    iDataError = 1;
    MessageBox("Not all data received in allotted time.", "Error",
MB_OK+MB_ICONERROR);
    break; // terminate do loop if Sleep has been run 300 times
    }
   Sleep(100); // 100 ms delay, required!
   //TRACE("\nOnRx3\tiMaxDelay = %d", iMaxDelay);
   dBytesWaiting = spPort.BytesWaiting();

if( (int) dBytesWaiting > iBufSize ) dBytesWaiting = (DWORD) iBufSize;
   iBytesWaiting = (int) dBytesWaiting;
   spPort.Read(ucData, dBytesWaiting);
```

```
//TRACE("\nOnRx3b\tiBytesWaiting = %d", iBytesWaiting);
     if(iDataSize+iBytesWaiting >= iBmpSize) {
       iBytesWaiting = iBmpSize - iDataSize;
       iDataFinish = 1; // get out of do loop
       }
     //TRACE("\nOnRx3c\tucData = %s", ucData);
     memcpy( &ucDataSorted[iDataSize], ucData, iBytesWaiting*sizeof(unsigned char) );
     iDataSize += iBytesWaiting;
     //TRACE("\nOnRx4\tiDataSize = %d", iDataSize);
     //TRACE("\nOnRx5\tucDataSorted = %s", ucDataSorted);
     } while (!iDataFinish);
   }
  //TRACE("\nOnRx6\tucDataSorted = %s", ucDataSorted);

  if(!iDataError) {
    ShowBitmap2(ucDataSorted);
    iFlagDataReceived = 1; // tell OnPaint to redraw 2nd frame
    }
}
void CSendBMPDlg::ShowBitmap2(unsigned char *pCharPixel)
//unsigned char *pCharPixel2 = new unsigned
char[m_bmpBM.bmWidth*m_bmpBM.bmHeight*4];
  //unsigned char *pCharPixel2 = new unsigned
char[m_bmpBM.bmWidth*m_bmpBM.bmHeight];
  //int i, j;
```

```
        CDC *pdc = m_ctlFrame2.GetDC(); // device context for painting
 CRect lRect;
 CDC dcMem;

 //  without clone BMFIH and BMFH headers, we need to create 32-bit bitmap data to
be
 //  compatible with screen display resolution ourselve!
 /*if(m_bmpBM.bmBitsPixel == 24) {
  for(i=0, j=0; i<m_bmpBM.bmWidthBytes*m_bmpBM.bmHeight; i+=3, j+=4) {
   pCharPixel2[j] = pCharPixel[i];
   pCharPixel2[j+1] = pCharPixel[i+1];
   pCharPixel2[j+2] = pCharPixel[i+2];
   pCharPixel2[j+3] = 255;
   }
  }
 else if(m_bmpBM.bmBitsPixel == 8) {
  for(i=0, j=0; i<m_bmpBM.bmWidthBytes*m_bmpBM.bmHeight; i++, j+=4) {
   pCharPixel2[j] = pCharPixel[i];
   pCharPixel2[j+1] = pCharPixel[i];
   pCharPixel2[j+2] = pCharPixel[i];
   pCharPixel2[j+3] = 255;
   }
  }
```

```
 m_bmpBitmap2.CreateCompatibleBitmap(pdc, m_bmpBM.bmWidth,
m_bmpBM.bmHeight);

 m_bmpBitmap2.SetBitmapBits(m_bmpBM.bmWidth*m_bmpBM.bmHeight*4,
pCharPixel2);*/


 // no need to change m_bmpBitmap2 if ShowBitmap2 is called by OnPaint

 if(pCharPixel != NULL)

  m_bmpBitmap2.SetBitmapBits(m_bmpBM.bmWidthBytes*m_bmpBM.bmHeight,
pCharPixel);


 //delete[] pCharPixel;


 // Create a device context to load the bitmap into

 dcMem.CreateCompatibleDC(pdc);

 // Select the bitmap into the compatible device context

 //CBitmap *pOldBitmap = (CBitmap *) dcMem.SelectObject(m_bmpBitmap2);

 dcMem.SelectObject(m_bmpBitmap2);

 // Get the display area available

 m_ctlFrame2.GetClientRect(lRect);

 lRect.NormalizeRect();
```

```
// Copy and resize the bitmap to the dialog window
  pdc->StretchBlt(0, 0, lRect.Width(), lRect.Height(), &dcMem, 0, 0,
m_bmpBM.bmWidth,
              m_bmpBM.bmHeight, SRCCOPY);

  //dcMem.SelectObject(pOldBitmap);
  m_ctlFrame2.ReleaseDC(pdc);
}

void CSendBMPDlg::OnClear()
{
            // TODO: Add your control notification handler code here

}
```

APPENDIX B

Reference Coding for Sorting Characters

```
        // SerialNiosDlg.cpp : implementation file
//


#include "StdAfx.h"
#include "SerialNios.h"
#include "SerialNiosDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


// CSerialNiosDlg dialog


CSerialNiosDlg::CSerialNiosDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CSerialNiosDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CSerialNiosDlg)
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
  }
```

```
	void CSerialNiosDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSerialNiosDlg)
	DDX_Control(pDX, IDC_EDIT2, m_ctlOutputStr);
	DDX_Control(pDX, IDC_EDIT1, m_ctlInputStr);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CSerialNiosDlg, CDialog)
	//{{AFX_MSG_MAP(CSerialNiosDlg)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDSEND, OnSend)
	ON_BN_CLICKED(IDEXIT, OnExit)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
// CSerialNiosDlg message handlers

BOOL CSerialNiosDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
```

```
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	return TRUE;  // return TRUE  unless you set the focus to a control
}
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSerialNiosDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
```

```
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;
                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}
// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSerialNiosDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}
#include "EasyD12.H"
#pragma comment (lib, "EasyD12.lib")
void CSerialNiosDlg::OnSend()
{
        CString strNiosInput;
        m_ctlInputStr.GetWindowText(strNiosInput);
        strNiosInput += "#";
```

```
iStringSize = strNiosInput.GetLength();

        BYTE *p = (BYTE*) strNiosInput.GetBuffer(0);
        WritePort1( (BYTE *) p, iStringSize);

        strNiosInput.ReleaseBuffer();

        char *pBuffer = new char [iStringSize + 1];
        ReadPort2( (BYTE *) pBuffer, iStringSize);
        pBuffer[iStringSize] = '\0';
        m_ctlOutputStr.SetWindowText(pBuffer);
        delete[] pBuffer;

        //// TODO: Add your control notification handler code here
 // CString strNiosInput;

 // /*void Open(int nPort, DWORD dwBaud=9600, Parity parity=NoParity, BYTE
DataBits=8,
 //         StopBits stopbits=OneStopBit, FlowControl fc=NoFlowControl,
 //         BOOL bOverlapped=false);*/
 // spPort.Open(1, 115200, CSerialPort::NoParity, 8, CSerialPort::OneStopBit,
 //         CSerialPort::NoFlowControl, false);
```

```
		// m_ctlInputStr.GetWindowText(strNiosInput);
 // iStringSize = strNiosInput.GetLength();
 // //DWORD Write(const void *lpBuf, DWORD dwCount);
 // spPort.Write(strNiosInput, (DWORD) iStringSize);
 // spPort.TransmitChar('\r'); // send CR to comply with scanf protocol in Nios/Nios II
 // OnRx();
}
void CSerialNiosDlg::OnExit()
{
		// TODO: Add your control notification handler code here
  spPort.Close();
  OnOK();
}
void CSerialNiosDlg::OnRx()
{
  CString strNiosOutput = "";
  CString strSortedOutput = "";
  unsigned char ucData[400] = "";
  int iMaxDelay = 0;
  enum State {NoData, DataBegin, DataFinish, DataError};
  State eState = NoData;
  DWORD dBytesWaiting;
  int iBytesWaiting;
```

```
Sleep(100);  // wait 100 ms
 //DWORD BytesWaiting(); returns # of bytes waiting to be read from serial port
 dBytesWaiting = spPort.BytesWaiting();
 iBytesWaiting = (int) dBytesWaiting;
 //TRACE("\nOnRx1\tiBytesWaiting = %d", iBytesWaiting);
 //DWORD Read(void *lpBuf, DWORD dwCount);
 spPort.Read(ucData, dBytesWaiting);
 strNiosOutput = ucData;
 //TRACE("\nOnRx1a\tstrNiosOutput = %s", strNiosOutput);

 //  Not required for Nios II. ////////////////////////////////////////////////////
 /*  scanf in Nios echos whatever it receives! Hence we need protocol to identify
    beginning of data actually sent by Nios. The string before actual data is "!#".
    So search for this string and capture the data following it.  */
 /*int iIndexStart;
 while (eState == NoData) { // while loop
  do {
    // returns -1 if char is not in CString, otherwise returns position of char
    iIndexStart = strNiosOutput.Find('!');
    //TRACE("\nOnRx2\tiIndexStart = %d\n", iIndexStart);
    if(iIndexStart == -1) break; // no '!', so get out of do loop
    else {  // data has '!'
     if(strNiosOutput[iIndexStart+1] == '#') { // next data after '!' is '#'?
       // delete characters up to '#'
```

```
strNiosOutput.Delete(0, iIndexStart+2);
        //TRACE("\nOnRx2a\tstrNiosOutput = %s", strNiosOutput);
        eState = DataBegin; // receives "!#", hence exit while loop
        break;  // get out of do loop
        } // next data after '!' is '#'?
       else strNiosOutput.Delete(0, iIndexStart+1); // next data after '!' is not '#'
       }  // data has '!'
     } while (1);
   if(iMaxDelay++ == 50) { // terminate while loop if Sleep has been run 50 times
     eState = DataError;
     MessageBox("Nios fails to send data in allotted time.", "Error",
MB_OK+MB_ICONERROR);
     break; // get out of while loop
     }
   if(eState == NoData) {
     Sleep(100);  // wait 100 ms
     dBytesWaiting = spPort.BytesWaiting();
     iBytesWaiting = (int) dBytesWaiting;
     //spPort.Read(&strNiosOutput, dBytesWaiting);
     spPort.Read(ucData, dBytesWaiting);
     strNiosOutput = ucData;
     //TRACE("\nOnRx2b\tstrNiosOutput = %s", strNiosOutput);
     }
   } // while loop*/
```

```
strSortedOutput = strNiosOutput;
 // Grab data until data size equals the known data size.
 if( (strSortedOutput.GetLength() < iStringSize) && (eState != DataError) ) {
  iMaxDelay = 0;
  do {
   if(iMaxDelay++ == 10) { // terminate do loop if Sleep has been run 300 times
    eState = DataError;
    MessageBox("Not all data received in allotted time.", "Error",
MB_OK+MB_ICONERROR);
    break;
    }
   Sleep(100); // 100 ms delay, required!
   dBytesWaiting = spPort.BytesWaiting();
   iBytesWaiting = (int) dBytesWaiting;
//TRACE("\nOnRx3a\tiBytesWaiting = %d", iBytesWaiting);
   spPort.Read(ucData, dBytesWaiting);
   strNiosOutput = ucData;
   //TRACE("\nOnRx3b\tucData = %s", ucData);
   if(strSortedOutput.GetLength()+iBytesWaiting >= iStringSize) {
    iBytesWaiting = iStringSize - strSortedOutput.GetLength();
    eState = DataFinish;
    }
   strSortedOutput += strNiosOutput.Left(iBytesWaiting);
   } while (eState != DataFinish);
  }
```